Understanding Malware

2015/08/14 Security Camp 2015 13-D, 14-D  
JPCERT/CC Analysis Center  
You NAKATSURU
Notice

These training materials are used for "Security Camp 2015" in Japan
   — Security training program for students to discover & nurture young talent
   — https://www.ipa.go.jp/jinzai/camp/ (Japanese only)

The training course consists of the following 2 parts
   — Malware, Malware analysis basics, Static analysis basics
     ▪ Learning basic knowledge for malware analysis
     ▪ Malware analysis
     ▪ Understanding details of malware samples using static analysis method

The training mainly focuses on 32bit Windows malware

Some slides have display problems due to animation

Any questions and comments are welcome
   — Please contact us at aa-info@jpcert.or.jp
Agenda

- Basic Knowledge
- Malware Analysis
  - Simple HTTP Bot
  - Banking Trojan
- Bonus
  - Shellcode
  - MWS Cup
- Discussion
Objectives of This Session

Understanding malware

- Windows features used by malware
- Implementation of "real" malware
  - HTTP Bot
  - Banking Trojan

Understanding static analysis

- Difficulties and Challenges
Windows Malware Analysis
(recap) Malware Analysis Flow

Start

Surface analysis
- enough?
  - Yes
  - No

Runtime analysis
- enough?
  - Yes
  - No

Static analysis
- Summarize Result
- End
## (recap) Analysis Process Comparison

<table>
<thead>
<tr>
<th>Overview</th>
<th>Surface analysis</th>
<th>Runtime analysis</th>
<th>Static analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Retrieve surface information from targets <strong>without execution</strong></td>
<td><strong>Execute samples</strong> and monitor its behavior</td>
<td><strong>Read codes</strong> in binary files and understand its functionality</td>
</tr>
<tr>
<td>Output</td>
<td>- Hash values</td>
<td>Activity of - File system - Registry - Process - Network</td>
<td>Malware’s functionality e.g. - Bot commands - Encode/decode methods</td>
</tr>
<tr>
<td></td>
<td>- Strings</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- File attributes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Packer info</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Anti-virus detection info</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security risk</td>
<td><strong>Low</strong></td>
<td><strong>High</strong></td>
<td><strong>Moderate</strong></td>
</tr>
<tr>
<td>Analysis coverage</td>
<td>Low</td>
<td><strong>Moderate</strong></td>
<td><strong>High</strong></td>
</tr>
</tbody>
</table>
## Static Analysis Tools

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disassembler</td>
<td>IDA</td>
<td>Disassembles more than 50 architectures</td>
</tr>
<tr>
<td>Decompiler</td>
<td>Hex-rays</td>
<td>x86/ARM binary to C source code</td>
</tr>
<tr>
<td>Decompile</td>
<td>VB</td>
<td>Visual Basic binary to Visual Basic source code</td>
</tr>
<tr>
<td></td>
<td>.NET Reflector</td>
<td>.NET binary to .NET source code</td>
</tr>
<tr>
<td>Debugger</td>
<td>OllyDbg</td>
<td>World famous X86 debugger</td>
</tr>
<tr>
<td></td>
<td>Immunity</td>
<td>Python familiar x86 debugger</td>
</tr>
</tbody>
</table>
BASIC KNOWLEDGE
PE (Portable Executable) File Format


- Consists of headers and multiple sections, will be extended on memory
  - Header: File Information
    - Entry point
    - Timestamp
    - Section’s info
    - etc.
  - Section: Byte code, data

PE header

Section 1 .text (code)

Section 2 .rdata (data)

Section 3 .data (data)
"EXE" and "DLL" are 2 most common file types in PE (Portable Executable) file format
— "Characteristics" of PE header

<table>
<thead>
<tr>
<th></th>
<th><strong>EXE</strong></th>
<th><strong>DLL</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Format</strong></td>
<td>Portable Executable</td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>Independent application file</td>
<td>Collection of functions as shared library</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>explorer.exe, iexplore.exe</td>
<td>kernel32.dll, shell32.dll</td>
</tr>
</tbody>
</table>
| **Execute timing** | • Main function  
  • when the file is executed | • Main function  
  • when the DLL is loaded/unloaded  
  • when a thread start/exit  
  • Exported function  
  • when is called |
Process & Virtual Memory

- 4GB per process (32bit Windows)
  - User space 2GB
    - allocated for each process, able to access each other
  - Kernel space 2GB
    - shared with all processes

**User space**
- Application code
- Various DLL code
- Stack, Heap
- etc.

**Kernel Space**
- Windows kernel
- Device driver
- Page pool
- etc.
Finding Main Function

Windows executable binary file will be started with initial processing to launch the process.

To find main function

Understand its initialization routine

- Compile & Disassemble your program

Use tools

- OllyDbg / Immunity Debugger
- IDA Starter/Pro

Use your sixth sense

- Based on your experience
Important Points

Do not read everything

- Time is money
- Remember "efficient code analysis"

Analysis is not our purpose

- Wrap up analysis results for incident response/information sharing
  - Analysis report/note
  - Commented IDB file
Malware Analysis

LET'S ANALYZE SIMPLE HTTP BOT
A kind of HTTP bot spread through mass emails

malware_sample1.idb
Exercise 1. Malware Analysis

i. Describe the following points of the target
   — Details of each bot command
   — Decode method
   ■ Try to decode malware_sample1_data.bin

ii. Make your IDB
    — Fill in information that you analyzed
# Point 1. AutoRun Function

- Want to launch after rebooting the OS
  - Copy itself into start up folder
  - Add a registry entry to AutoRun part

## Registry entries related to AutoRun

<table>
<thead>
<tr>
<th>Registry Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKCU\SOFTWARE\Microsoft\Active Setup\Installed Components</td>
<td></td>
</tr>
<tr>
<td>HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\Run</td>
<td></td>
</tr>
<tr>
<td>HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell</td>
<td></td>
</tr>
<tr>
<td>HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run</td>
<td></td>
</tr>
<tr>
<td>HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce</td>
<td></td>
</tr>
<tr>
<td>etc.</td>
<td></td>
</tr>
</tbody>
</table>
Point 2. Hiding API name

Getting API address using GetProcAddress

```
push    offset aInternetCloseh ; "InternetCloseHandle"
push    offset ModuleName     ; "wininet.dll"
call    ds:GetModuleNameA
push    eax            ; hModule
call    ds:GetProcAddress
push    esi            ; hModule
```
Point 3. HTTP Communication

There are many ways to communicate using HTTP

**WinINet APIs**
- InternetOpen, HttpSendRequest, ...

**WinSock APIs**
- socket, connect, send, recv, ...

**WinHTTP APIs**
- WinHttpConnect, WinHttpSendRequest, ...

**etc.**
- URLDownloadToFile, ...

Point 4. Encoding (Obfuscation)

- Encode (encrypt) data to avoid being easily found
  - Strings stored in the binary
    - File name, Registry entry name, Server address
  - Packet

- Various methods are available

<table>
<thead>
<tr>
<th>Method</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>xor (exclusive or)</td>
<td>'a' ^ 0x05 = 'd'</td>
</tr>
<tr>
<td>ror/rol (rotate right/left)</td>
<td>rol 'a', 1 = 0xC2</td>
</tr>
<tr>
<td>base64</td>
<td>-</td>
</tr>
<tr>
<td>RC4</td>
<td>-</td>
</tr>
<tr>
<td>AES</td>
<td>-</td>
</tr>
</tbody>
</table>
Point 4. Encoding (Obfuscation)

- e.g. HTTP packet obfuscation
  - Data encoded using "xor" or "ror/rol" may become non-ASCII
  - Combination with base64 encoding is a common approach

```
make text data
  xor/ror/rol encode
    base64 encode
  binary data
    text data
  Send data
```
Point 5. Bot command

Bots are capable to communicate with C&C servers to get commands to work

- Receive command
- Match
- Compare
- Information theft
- Match
- Compare
- Download & exec
- Match
- Compare
- Key logging
- Match
- Compare
- Remote shell
- Match
- Send results
Exercise 1. Malware Analysis

i. Describe the following points of the target
   — Details of each bot command
     ■ "upload": Download file from arbitrary URL
     ■ "uploadexec": Download & execute file
     ■ "xxx": Execute arbitrary shell command (Remote shell)
     ■ "xxxx": Upload specific file to C&C server
   — Decode method
     ■ Try to decode malware_sample1_data.bin
     ■ Wide char -> Multi byte char -> xor 0x53

ii. Make your IDB
   — Fill in information that you analyzed
Malware Analysis

LET'S ANALYZE BANKING TROJAN
Analysis Target

- Dropper + Tinba

Diagram:
- Drive-by-Download attack
- malware_sample2.idb
- malware_sample3.idb
Exercise 2. Malware Analysis

i. Analyze position independent data addressing in "malware_sample3.idb"

ii. Analyze "malware_sample_clean.idb" and describe the following points of the target
   — How to avoid anti runtime analysis technique
   — Installation flow
   — Target web browser

iii. Make your IDB
   — Fill in the information that you analyzed
Point 1. Dropping Files

- Create another file
  - Dropped files usually contains the main function for the attack

- 2 common methods

  **Download from the server**
  - Downloader

  **Store drop files in programs**
  - Data / resource / overlay
  - Usually encoded
Point 1. Dropping Files

Dropping file from resource

1. Find and load the encoded data from resources
   - FindResource
   - LoadResource
   - SizeofResource
   - LockResource

2. Decode
   - HeapAlloc
   - RtlDecompressBuffer

3. Write decoded data to the file
   - CreateFile
   - WriteFile
   - CloseHandle

main.exe
Point 2. Position Independent Data Addressing

Push strings using CALL instruction

Push address of "ntdll" & jump to next instruction
Point 3. Anti Runtime Analysis

Some types of malware are clever enough to detect analysis activity

— To avoid analysis by malware analysts
Point 4. Code Injection

Method to execute arbitrary code in another process

Malware

malware.exe

DLLs

Target

target.exe

code

DLLs

Open the target process

Allocate

Write

Create remote thread
Point 5. API Hooking

- Method to execute arbitrary code when API is called
  - Logging/Modifying parameters passed to APIs

Before

```
+-------------------+
| target.exe        |
+-------------------+
| IAT               |
+-------------------+
| DLLs              |
+-------------------+
```

After

```
+-------------------+
| target.exe        |
+-------------------+
| IAT               |
+-------------------+
| code              |
+-------------------+
| DLLs              |
+-------------------+
```
Exercise 2. Malware Analysis

i. Analyze position independent data addressing in "malware_sample3.idb"

ii. Analyze "malware_sample_clean.idb" and describe the following points of the target
   — How to avoid anti runtime analysis technique
     ■ Mouse cursor checking, Disk cylinder checking
   — Installation flow
     ■ See "aa_install_as_speechengines" function
   — Target web browser
     ■ Internet Explorer, Firefox, Chrome, Maxthon

iii. Make your IDB
    — Fill in the information that you analyzed
Bonus:
Shellcode Analysis
BASIC KNOWLEDGE
(recap) Exploiting Vulnerability

- Buffer overflow, etc.
- Take control and execute arbitrary code
- Shellcode for malware execution
- Malware
What Shellcode is

- Code snippet that is executed after exploiting
  - e.g. Stack based buffer overflow + Heap spray
## Comparison With Executable File

<table>
<thead>
<tr>
<th></th>
<th>Executable file</th>
<th>Shellcode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Format</strong></td>
<td>PE file format (header, code, data, etc.)</td>
<td>Code only</td>
</tr>
<tr>
<td><strong>Load address</strong></td>
<td>Specified at PE header</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>API address</strong></td>
<td>PE loader will resolve API address</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Shellcode has some routines to retrieve these addresses.
Basic Process of Shellcode

1. Specify its base address
2. Decode its main part
3. Get API addresses
4. Drop malware
   - Stored in the file
   - Download
5. Execute malware
TIB & PEB

Thread Information Block (TIB)

- Also called "Thread Environment Block (TEB)"
- Contains thread related information
  - Thread context, PEB, etc.

Process Environment Block (PEB)

- Contains process related information
  - PID, Loaded modules, etc.

Used by shellcode to resolve API address
TIB in Segment Register

- FS register points to TIB
Loading to IDA

- Load as a 32bit code

- Recommendation
  - Change loading offset to 0x00010000 to avoid analysis failure (in some cases)
Shellcode Analysis

LET'S ANALYZE
Analysis Target

- Shellcode cropped from memory dump

![Diagram showing memory, stack, application, and shellcode with "bonus_sc.idb" highlighted]
Point 1. Getting Base Address

To calculate relative address

Jump to next instruction
• call $+5

Get return address from stack
• pop ebp

Calculate base address
• sub ebp, 0Dh
Point 2. GetProcAddress

Step 1: getting base address of kernel32.dll

PEB
• fs:[eax+30h]

Ldr
• [eax + 0Ch]

InInitializationOrderModuleList
• [eax+1Ch]
Point 2. GetProcAddress

Step 2: parsing DLL file to get API address

- \[ +3\text{Ch} \]
- \[ +3\text{Ch} +78\text{h} \]
- Stored address of Export Directory at the head
Point 2. GetProcAddress

2 methods to obtain API addresses

Get all API address manually

• Parse DLL file every time
  • Compare export function name with API to use

Use GetProcAddress

• Use GetProcAddress after getting address of GetProcAddress
FYI. API Hashing

Recent shellcode use hash value of API name for anti-virus/analysis

```
push    ebx
push    esi
push    edi
push    0D5786h     ; kernel32.d1!LoadLibraryA
push    0D4E88h
call    aa_get_proc_address_from_hash
mov     [ebp+var_4], eax
push    348BFAh     ; kernel32.d1!GetProcAddress
push    0D4E88h
call    aa_get_proc_address_from_hash
mov     [ebp+var_8], eax
jmp     loc_100013F
```

Discussion
Questions?